Application and comparison of multiple approaches on game 2048

Zhengxian Lin[1] and Zhiyuan He[2]

***Abstract*— This report mainly introduces multiple reinforcement learning approaches (Q-learning, SARSA, Mone-Carlo On-Policy Control, Linear Approximation) and one heuristic search algorithm (UCT) on game 2048. Also, experiment and analyze the performance of these approaches.**

I. INTRODUCTION

Artificial intelligence has been recently widely used in strategic game systems for its outstanding learning and problem-solving skills. However, there are various approaches in the artificial intelligence field. Choosing an appropriate approach can be tricky.

This report mainly introduces multiple common approaches used in the artificial intelligence field and experiments them with a game named 2048. Section 2 introduces the game 2048 and these approaches, including how to adjust them for the game 2048. Section 3 explains how the experiment is set and present the results of the experiment. Section 4 analyses and compares the experiment results of these different approaches. Section 5 presents the things we learned form this project.

II. GAME AND APPROACHES

A. Game 2048

The game 2048 is a game played on a 4×4 grid with numbered tiles that slide to four directions which are up, down, left and right. The interface is shown in Fig.1.

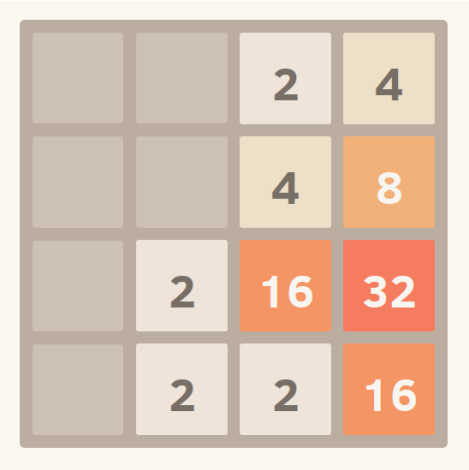

Fig. 1. The interface of 2048

[1]Zhiyuan He, Computer Science, Oregon State University. Email: hezhi@oregonstate.edu
[2]Zhengxian Lin,Computer Science, Oregon State University. Email: linzhe@oregonstate.edu

Every turn, a new tile will randomly appear in an empty spot on the board with a value of either 2 or 4. Tiles slide as far as possible in the chosen direction until they are stopped by either another tile or the edge of the grid. If two tiles of the same number collide while moving, they will merge into a tile with the total value of the two tiles, and the total score is incremented by this total value. The resulting tile cannot merge with another tile again in the same move. When the player has no legal moves (there are no empty spaces and no adjacent tiles with the same value), the game ends. As shown in Fig.2.

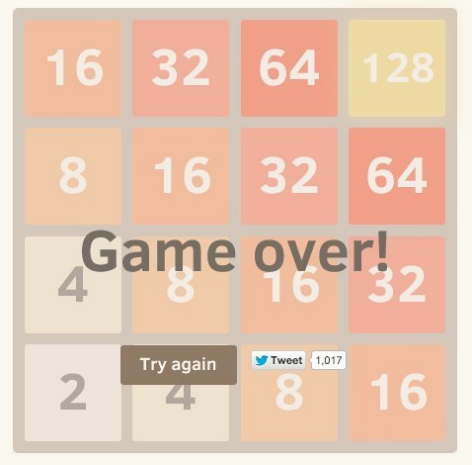

Fig. 2. The interfave of 2048 game over

The game 2048 has two characters which make it hard to solve. One character is the enormous state spaces. The biggest tile can be got from this game is 2^{17} as there are only 16 spots. And for each spot, it can be 2, 2^2, 2^3, ..., 2^{17} and empty, 17 possibles. So there are 18^{16} states for this game. Another character is stochastic. A new tile with value of 2 or 4 will random appear in an empty on the board. This stochastic rule adds some uncertainties in this game and makes the game harder to solve.

B. Q-learning

Q-learning is a reinforcement learning technique. It is an off-policy learning algorithm which means it learns an optimal policy regardless of the behavior policy being followed.

Q-learning is a TD method. It use the notation $Q(s, a)$ to denote the value of doing action a in state s. It learns a Q-function does not need a model of the form $P(s, a, s')$, either for learning or for action selection. For this reason, Q-learning is also called a model-free method.

In original Q-Learning, there is a limitation, that is, it cannot handle the problem which has enormous state space, like game 2048. Thus, we adjust the original Q-Learning to fit our game, a Q-Learning with step limitation for each step. Fig.3 shows the pseudocode.

Initialize: if Q is None:
Q arbitrary value function
Loop for each episode:
$s \leftarrow copy(gameboard)$
// Get current state of new episode
While s **is not terminal and steps <limitedSteps:**
$legalAactionSet = legalAction(s)$
$a \leftarrow explore/exploitpolicy(s, legalAactionSet)$
$r \leftarrow takeAction(a)$
$s' \leftarrow copy(gameboard)$
// Choose a via explore/exploit policy based on Q Take action a and observe r and s
$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \beta \max_{a' \in A} Q(s',a') - Q(s,a))$
$s \leftarrow s'$

Fig. 3. Pseudocode of Q-learning

The reason why we do this adjust is the longer step cause larger state space. In our case, the upper of a number of step is 2^{18}, which is very large. So, setting a limitation of the future step can be reasonable to make the trade-off between considering the future and the number of states. In our case, the limited steps are 10, 15, 20, 50 and 100. With these limited steps, the state space is much smaller.

C. SARSA

Like Q-learning, SARSA (for State-Action-Reward-State-Action) is also a reinforcement learning technique and TD method. The difference from Q-learning is quite subtle: whereas Q-learning backs up the best Q-value from the state reached in the observed transition, SARSA waits until an action is actually taken and backs up the Q-value for that action.

Unlike the Q-learning which learns an optimal policy regardless of the behavior policy being followed, SARSA is an on-policy learning algorithm, it learns by trying to evaluate the policy being followed during learning.

For SARSA, there is still same problem, which is enormous state space, as same as Q-Learning, we also do the same adjustment for it. Fig.4 shows the pseudocode.

D. Monte-Carlo On-Policy Control

Monte-Carlo On-Policy Control is a generalized policy iteration based on policy learning technique. It continually trying to evaluate the Q-function of current behavior policy and continually adapt behavior policy to better policy based on Q-function.

For Monte-Carol On-Poilicy Control, we also same like Q-Learning and SARSA, which is setting limitation for simulation steps. Fig.5 shows the pseudocode.

Initialize: if Q is None:
Q arbitrary value function
Loop for each episode:
$s \leftarrow copy(gameboard)$
$legalAactionSet = legalAction(s)$
$a \leftarrow explore/exploitpolicy(s, legalAactionSet)$
// Get current state and action of new episode
While s **is not terminal and steps <limitedSteps:**
$r \leftarrow takeAction(a)$
$s' \leftarrow copy(gameboard)$
$legalAactionSet' = legalAction(s')$
$a' \leftarrow explore/exploitpolicy(s', legalAactionSet')$
// Choose a' via explore/exploit policy based on Q
$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \beta Q(s',a') - Q(s,a))$
$s \leftarrow s'$
$a \leftarrow a'$

Fig. 4. Pseudocode of SARSA

Initialize: If Total(s, a) is None:
$Total(s,a) \leftarrow 0$, $Count(s,a) \leftarrow 0$ for each (s,a)
Loop for each episode:
Generate episode of π**:** $s_0, a_0, r_1, ..., s_{T-1}, a_{T-1}, r_T$
// T is depended on terminal state and limited steps
FOR $t = 1$ to $T-1$,
IF s_t, a_t does not appear earlier in sequence THEN
$g_t = r_t + \beta r_{t+1} + ... + \beta^{T-t} r_T$
$Total(s_t, a_t) \leftarrow Total(s_t, a_t) + g_t$
$Count(s_t, a_t) \leftarrow Count(s_t, a_t) + 1$
$Q_\pi(s_t, a_t) = \frac{Total(s_t, a_t)}{Count(s_t, a_t)}$

Fig. 5. Pseudocode of Monte-Carlo On-Policy Control

E. Linear function approximation

Linear function approximation is a function using some feature of state and action to represent similar states, and it is always used for the problem which has the enormous state like game 2048. However, we have come out over ten features. The performance of this linear function is not good. It was learning very slightly. There are several important features we have come out:

1) The empty cell of the state
2) The number of merging
3) The Manhattan distance of each pairs of tiles with closest numbers. (which consider the distance between all tiles. This feature influences the future of this game)
4) The Monotonicity of each row and column (which is an important feature for this game)
5) The value of number of corner (which is we should consider badly at this game to avoid the max number affect other tiles due to it cannot be merge until last step to get new bigger number)
6) Each numbers value and position.

7) The relative position of each pairs of tiles with closest numbers, and the value of them.

We tried to combine all of them or several of them together and did Q-Learning agent, SARSA agent and MC agent with those linear function, but it does not work well. However, there is still a best one. The fig.6 showed the result of it which is using the No.7 feature only with MC agent. After learning, the average of it can reach 3 times score of baseline, which is random agents.

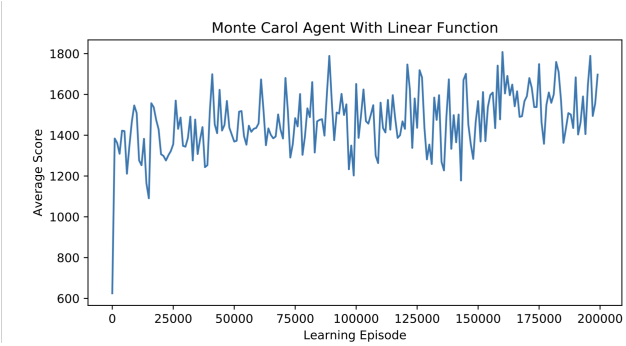

Fig. 6. MC with linear function approximation

We also try to figure out some possible reasons why the linear function does not work well. There are two possible reasons:

1) In game 2048, the good or bad state is not guided by the score of one step or several future steps. The actual score, as shown by the game, is not used to judge a state, since it is too heavily weighted in favor of merging tiles (when delayed merging could produce a large benefit), focusing on immediately merging everything actually leads to shorter games.
2) We cannot find out a good guidance that is strongly related to the longer future. We has try the number of steps until this state, the final score to updating all thetas of each state after end of game, but they are also failed.

F. UCT

Upper Confidence Bounds Trees (UCT) is one member of MCTS family. It combines UCB with MCTS. UCT is to calculate an actions value approximately through simulation. The roll out way we used in this paper is playing randomly. There are four steps for UCT:

First, selection: If the current node has been simulated(expanded) then using UCB to calculate its value by following expression:

$$\frac{w_i}{n_i} + C * \sqrt{\frac{\ln N_i}{n_i}}$$

- w_i is the number of wins.
- n_i is the number of plays.
- C is a constant parameter which is 45 in our project.
- N_i is the sum of the number of plays of all nodes which is at same layer with the current node.

Second, expansion: after selection, expands this node and get its children nodes.

Third, simulation: after expansion, random select a leaf node which has not value, which means which has not been simulated, doing simulation.

Forth, back propagation: after simulation, the current node state is the final state, go back to the current state which agent is playing now, and update all of the value of the nodes through the path.

For UCT, we also did some adjustment for it. The reason why we try to make these adjustments is the performance of UCT is not as good as we expected. After trying to run the original UCT, we found out that doing rollout until the end of the game will cause very high variance, which is the reason why the performance of UCT is not good.

As we said before, the upper number of steps of this game can be 2^{18}, so that the UCT cannot catch the key action. For this part, we set a boundary for roll out to decrease the variance, but in doing so, the bias of it is increased. The second adjustment is that we give some penalty for lost the game before the roll out reach the boundary because this is game have not win state, and lost latterly is better than lost early. Let assume that the deep of roll out is d, and the boundary is b, the penalty $p = 1/(d - b + 1)$, the roll out reward $r = r \times p$.

The presudo code is like follow:

Initialize: legalActionsSet = legalAction(gameboard)
Loop until reaching the number of simulation:
    $simulator \leftarrow createSimulator()$
    $s \leftarrow copy(gameboard)$
    $unActSet \leftarrow getUndoActions(gameboard)$
    $la \leftarrow legalActionsSet$
    while length of is 0 and simulator is not end
        $action \leftarrow UCBSeletedPolicy(s, la)$
        $backPropagationSet$ append $(s, action)$
        $simulator.takeAction(a)$
        $la \leftarrow legalAction(simulator.gameboard)$
        $unActSet \leftarrow getUndoActions(gameboard)$
    IF length of $unActSet$ **is not 0:**
        $ra \leftarrow randome(unActSet)$
        $backPropagationSet$ append $(s, action)$
        $simulator.takeAction(ra)$
    $reward \leftarrow simulator.simulation(randomPolicy)$
    $backpropagation(backPropagationSet, reward)$
Return best trial

Fig. 7. Pseudocode of UCT

After these adjustments, the score of UCT can be 5 times of before.

III. EXPERIMENTS

For the experiment, we choose the learning trial equals 100 and 500 and limited step equals 10, 15, 20, 50 and 100 to see the performance of all four approaches play 19 times.

The measurement of the performance is the score they got in each game and the average score of 19 times of play. The time of the experiment is quite as long as some approach may need hours to finish a play. And from the experiment, we expect to see the highest score for each approach can get and whether an approach might have a better average score. The relational of limited steps, the number of learning trials and the performance is also what we focus on.

IV. RESULTS

For all results, Q-Learning and SARSA use learning rate $\alpha = 0.1$, beta = 0.99, explore/exploit epsilon = 0.3. MC agent also use beta = 0.99, explore/exploit epsilon = 0.3. learning trails for Q-Learning, SASAR, and MC means episode, and for UCT, means the number of simulation.

A. Comparison of Single Play for all approaches

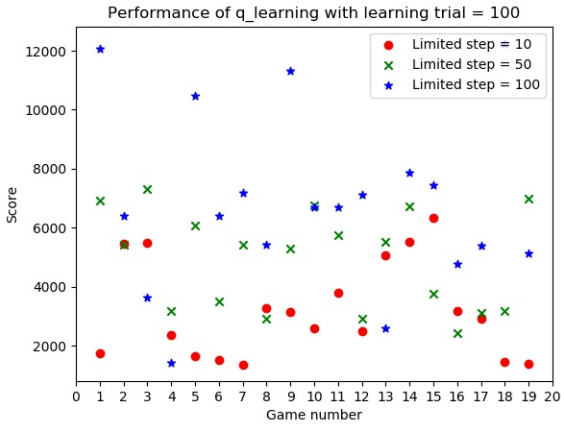

Fig. 8. The performance of Q-learning(learning trials=100) for 19 games

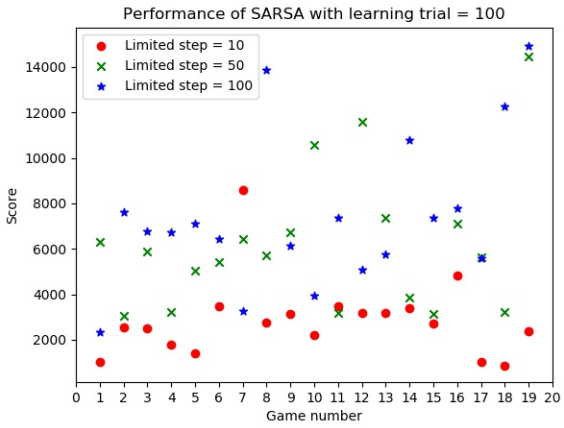

Fig. 9. The performance of SARSA(learning trials=100) for 19 games

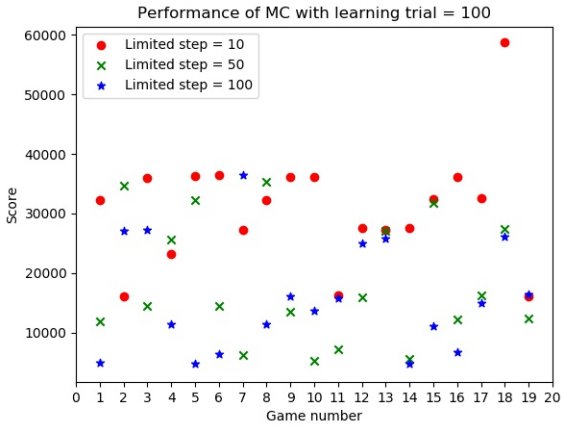

Fig. 10. The performance of MC(learning trials=100) for 19 games

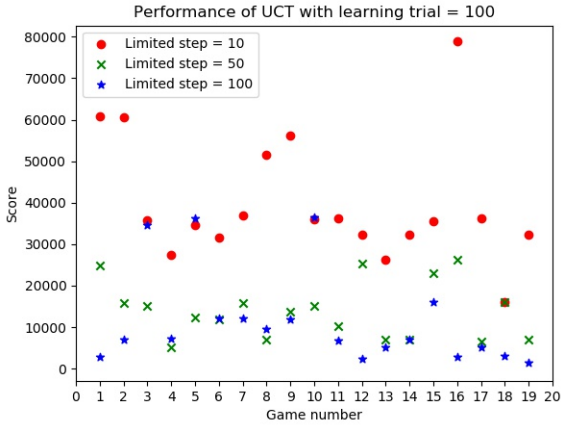

Fig. 11. The performance of UCT(learning trials=100) for 19 games

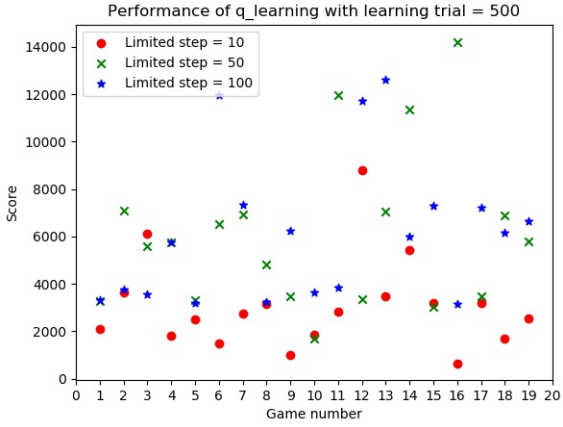

Fig. 12. The performance of Q-learning(learning trials=500) for 19 games

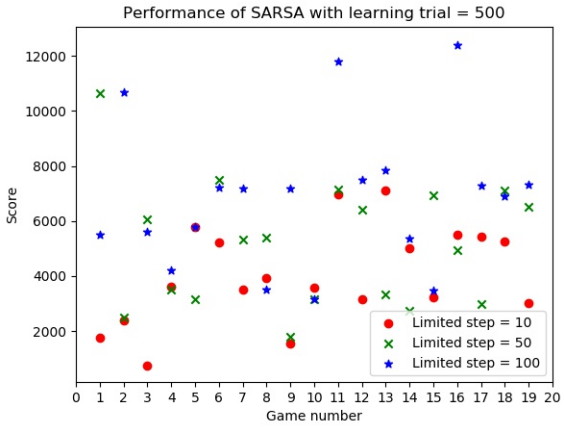

Fig. 13. The performance of SARSA(learning trials=500) for 19 games

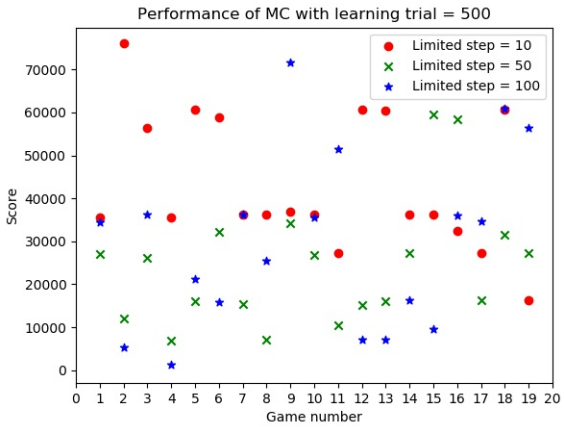

Fig. 14. The performance of MC(learning trials=500) for 19 games

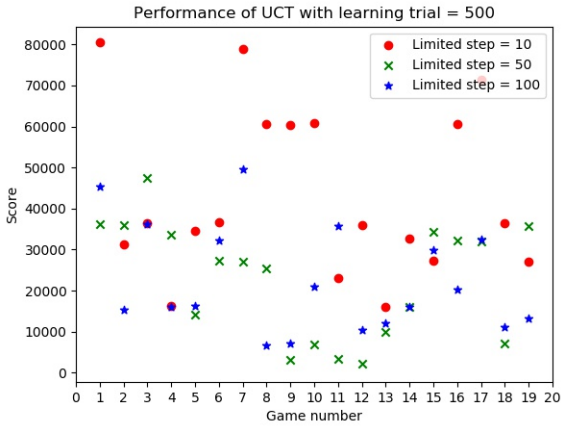

Fig. 15. The performance of UCT(learning trials=500) for 19 games

The fig.8 - fig.15 show the results of the score all approaches got in 19 plays. The learning trials are 100 and 500. From these results, we get the conclusions as follow:

1) The highest score is produced by UCT agent, and SARSA agent got the lowest score among these four agents. The interesting thing is both of them have same limited steps to compare the highest and lowest score.
2) The UCT agent and SARSA agent are more stable than Q-Learning and MC agent for this game.

B. Comparison of average score for each approaches

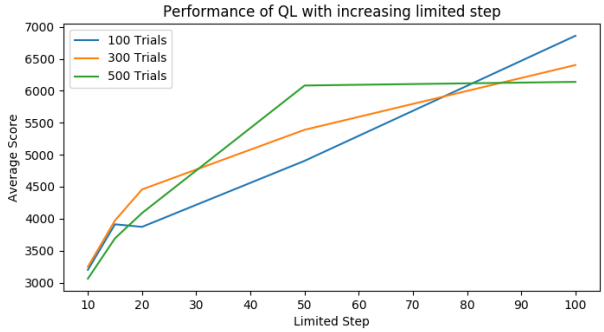

Fig. 16. The performance of Q-learning with increasing limited steps

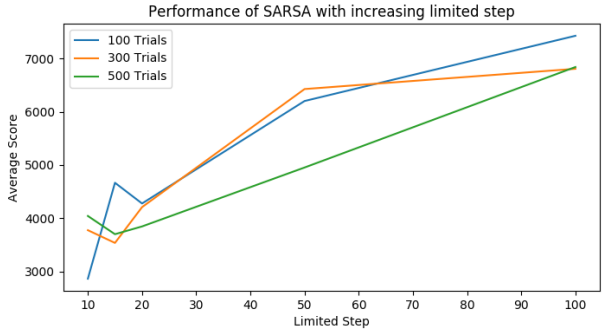

Fig. 17. The performance of SARSA with increasing limited steps

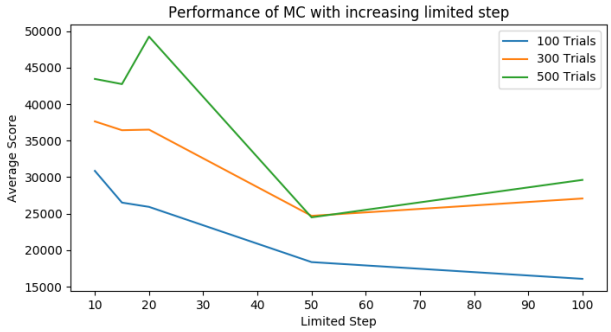

Fig. 18. The performance of MC with increasing limited steps

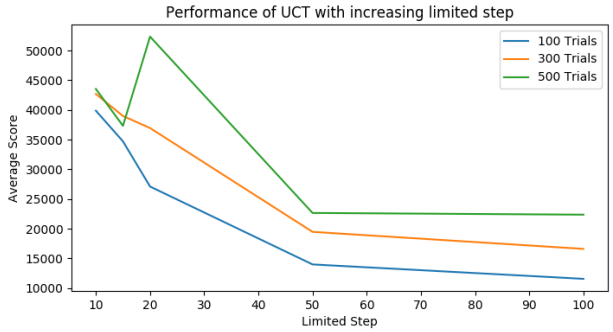

Fig. 19. The performance of UCT with increasing limited steps

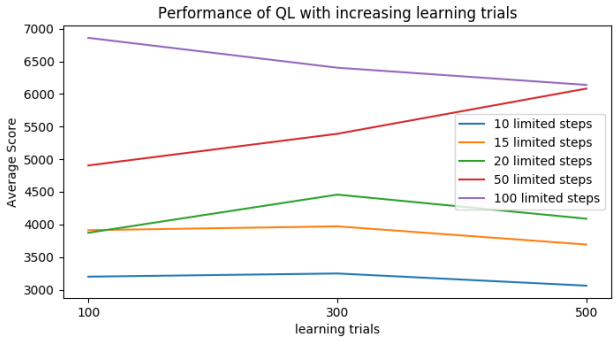

Fig. 20. The performance of Q-learning with increasing learning trials

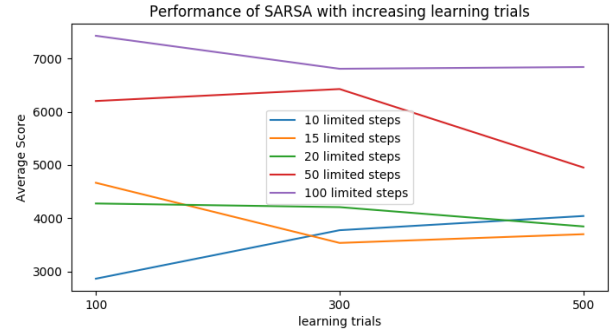

Fig. 21. The performance of SARSA with increasing learning trials

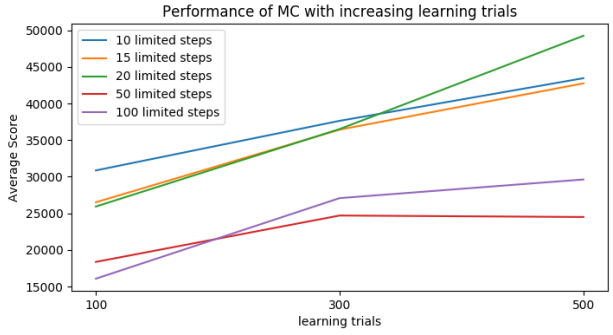

Fig. 22. The performance of MC with increasing learning trials

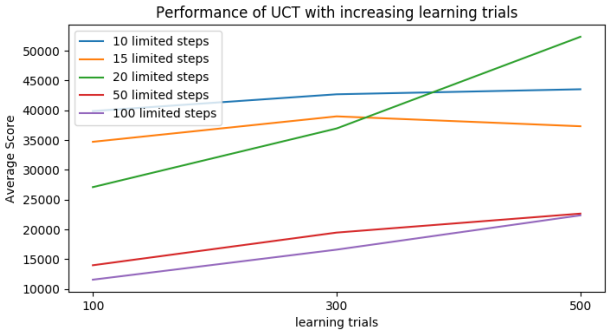

Fig. 23. The performance of UCT with increasing learning trials

The fig.16 - fig.23 show the results of each approach with different limited steps and learning trials. From these results, we get the conclusions as follow:

1) With the number of limited steps increased, the performance of UCT agent and MC agent are worse because of the higher variance. However, the performance of Q-Learning agent and SARSA agent are better, but not apparent.
2) With the number of learning trials increased, the performance of UCT agent and MC agent are better. However, the Q-Learning agent and SARSA agent are not influenced apparently by that.

C. Comparison among all approaches

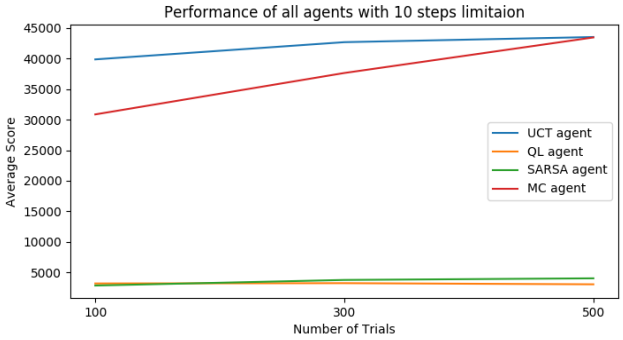

Fig. 24. The performance of all(limited steps = 10)

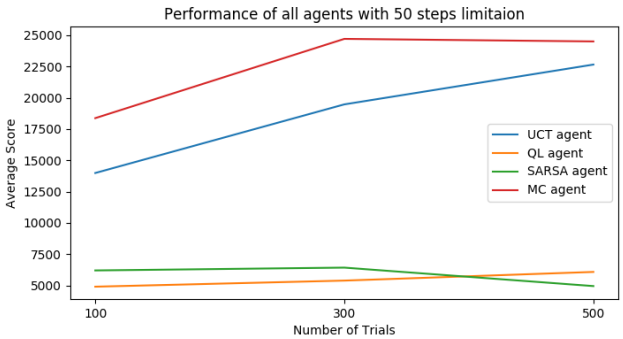

Fig. 25. The performance of all(limited steps = 50)

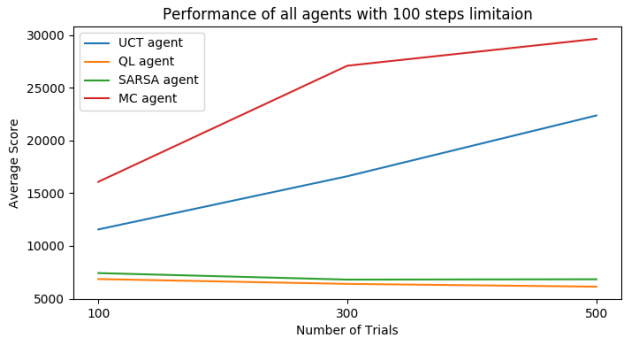

Fig. 26. The performance of all(limited steps = 100)

The fig.24 - fig.26 show the results of comparison among all approaches with different limited steps. From these results, we get the conclusions as follow:

1) The UCT agent and MC agent always much better than Q-Learning agent and SARSA agent. The possible reason is that the Q-Learning and SARSA are based on the temporal difference which considers one step reward badly, having a very high bias. As we said before, for this game, one step or several steps reward cannot guide a state to a better state, sometimes it is dangerous to follow the score. Thus, there is no surprise the performance of Q-Learning agent and SARSA agent is worse than the other two.
2) For MC agent and UCT agent, the former is affected by the number of limited steps less. When the limited step is small, the performance of UCT agent is better than MC, but when then limited step increase, like 50 and 100, the performance of MC agent is better than UCT agent.

The baseline of this project is a random agent which is average score is about 600.

Q-Learning agent and SARSA agent are like beginner player.

UCT agent and MC agent can reach the score that most of the players cannot make.

V. THINGS WE LEARNED

1) For different models MDP, several Reinforcement Learning methods can perform totally different. In the game 2048, the performance of Monte-Carol agent is very good, which we are surprised. Q-learning is bad, which we think it will be the best among three of them because we have seen how good it is in homework four.
2) Sometimes, adjustments of an algorithm for a specific problem is necessary. We can see the performance of UCT. After setting a boundary of the rollout, the performance can be over 5 times better than before. And, adjustment also can solve some tough problem, like the other three agents.
3) For an algorithm, we should try to understand deeply what makes it good or bad. For example, the Q-Learning and SARSA are based on the temporal difference equation which considers the one step reward badly, having a very high bias. Thus, for the problem which like game 2048, which agent should focus on very long future, they are not good at that.
4) For some problem, it is hard to use Linear function approximation to represent its state, especially when the state can grow up, which means there are new things came out in the future, like new number generated at game 2048.
5) Although the performance of UCT is good, it is too expensive to do many simulations for each step, especially when there are different legal actions for a different state, because it should get legal action every time.